A Prefetching Protocol for Streaming of Prerecorded Continuous Media in Wireless Environments

Frank H.P. Fitzek [a] and Martin Reisslein[b]

[a]Dept. of Electrical Eng., Technical University Berlin, 10587 Berlin, Germany
[b]Dept. of Electrical Eng., Arizona State University, P.O. Box 877206, Tempe, AZ 85287–7206

ABSTRACT

Streaming of continuous media over wireless links is a notoriously difficult problem. This is due to the stringent Quality of Service requirements of continuous media and the unreliability of wireless links. We develop a streaming protocol for the real–time delivery of prerecorded continuous media from a central base station to multiple wireless clients within a wireless cell. Our protocol prefetches parts of the ongoing continuous media streams into prefetch buffers in the clients. Our protocol prefetches according to a Join–the–Shortest–Queue policy. By exploiting rate adaptation techniques of wireless data packet protocols, the Join–the–Shortest–Queue policy dynamically allocates more transmission capacity to streams with small prefetched reserves. Our protocol uses channel probing to handle the location–dependent, time–varying, and bursty errors of wireless links. We evaluate our prefetching protocol through extensive simulations with VBR MPEG encoded video traces. Our simulations indicate that for bursty VBR video with an average rate of 64 kbit/sec and typical wireless communication conditions our prefetching protocol achieves client starvation probabilities on the order of 10^{-4} and a bandwidth efficiency of 90 % with prefetch buffers of 128 kBytes.

Keywords: Channel Probing, Prefetching, Prerecorded Continuous Media, Real–Time Streaming, Wireless Communication

1. INTRODUCTION

Due to the popularity of the World Wide Web retrievals from web servers are dominating today's Internet. While most of the retrieved objects today are textual and image objects, web–based streaming of continuous media, such as video and audio, becomes increasingly popular. At the same time there is increasingly the trend toward accessing the Internet and Web from wireless devices. The stringent Quality of Service (QoS) requirements of continuous media and the unreliability of wireless links combine to make streaming over wireless links a notoriously difficult problem. In this paper we develop a high performance streaming protocol for the real–time delivery of prerecorded continuous media over wireless links. We focus on the streaming in the downlink (base station to clients) direction. Our protocol allows for immediate commencement of playback. Our protocol gives a constant perceptual media quality at the clients while achieving a very high bandwidth efficiency. Our protocol achieves this high performance by exploiting two special properties of *prerecorded* continuous media: (1) the client consumption rates over the duration of the playback are known before the streaming commences, and (2) while the continuous media stream is being played out at the client, parts of the stream can be prefetched into the client's memory. The prefetched reserves allow the clients to continue playback during periods of adverse transmission conditions on the wireless links.

The prerecorded continuous media streams are prefetched according to a specific Join–the–Shortest–Queue (JSQ) policy, which strives to balance the prefetched reserves in the wireless clients within a wireless cell. The JSQ prefetch policy exploits rate adaptation techniques of wireless data packet protocols.[1] The rate adaptation techniques allow for the dynamic allocation of transmission capacities to the ongoing wireless connections. In the Code Division Multiple Access (CDMA) IS–95 (Revision B) standard, for instance, the rate adaptation is achieved by varying the number of codes (i.e., the number of parallel channels) used for the transmissions to the individual clients. Roughly speaking, the JSQ prefetch policy dynamically allocates more transmission capacity to wireless clients with small prefetched reserves while allocating less transmission capacity to the clients with large reserves. The ongoing streams

Further author information: (Send correspondence to M. Reisslein)
F. Fitzek: E-mail: `fitzek@ee.tu-berlin.de`, Web: `http://www-tkn.ee.tu-berlin.de/~fitzek`
M. Reisslein: E-mail: `reisslein@asu.edu`, Web: `http://www.eas.asu.edu/~mre`

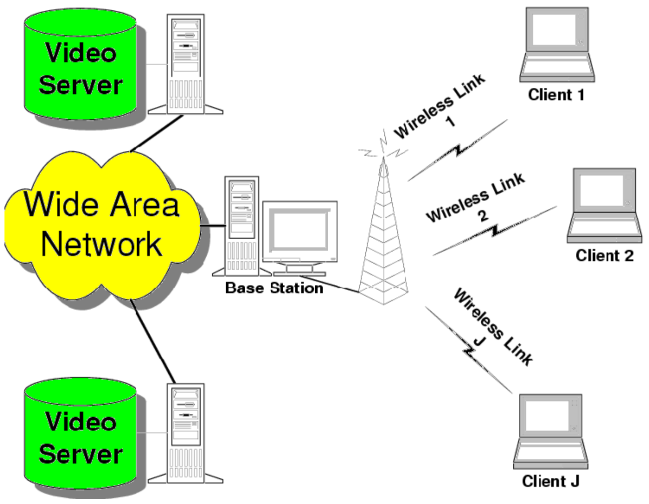

Figure 1. Architecture: A central base station streams prerecorded continuous media to wireless (and possibly mobile) clients within a wireless cell.

within a wireless cell collaborate through this lending and borrowing of transmission capacities. Channel probing is used to judiciously utilize the transmission capacities of the wireless links, which typically experience location–dependent, time–varying, and bursty errors. Our extensive numerical studies indicate that this collaboration is highly effective in reducing playback starvation at the clients while achieving a high bandwidth efficiency. For bursty VBR video with an average rate of 64 kbit/sec and typical wireless communication conditions our prefetching protocol achieves client starvation probabilities on the order of 10^{-4} and a bandwidth efficiency of 90 % with client buffers of 128 kBytes.

This paper is organized as follows. In Section 2 we describe our architecture for the streaming of prerecorded continuous media in the downlink direction. Our focus is on multi–rate CDMA systems. In Section 3 we develop the JSQ prefetching protocol. We also introduce channel probing; a mechanism that handles the location–dependent, time–varying, and bursty errors of wireless environments. We evaluate our prefetching protocol through extensive simulations. We discuss the related work in Section 4 and conclude in Section 5.

2. ARCHITECTURE

Figure 1 illustrates our architecture for continuous media streaming. A central base station provides streaming services to multiple wireless (and possibly mobile) clients within a wireless cell. Let J denote the number of clients serviced by the base station. We assume for the purpose of this study that each client receives one stream; thus there are J streams in process. The basic principle of our streaming protocol — exploiting rate adaptation techniques for prefetching — can be applied to any type of wireless communication system with a slotted Time Division Duplex (TDD) structure. The TDD structure provides alternating forward (base station to clients) and backward (clients to base station) transmission slots.

We initially consider a multi–code Code Division Multiple Access (CDMA) system, that is, a CDMA system that adapts rates for the synchronous transmissions in the forward direction by aggregating orthogonal code channels, that is, by varying the number of code channels used for transmissions to the individual clients. The second generation CDMA IS–95 (Rev. B) system is an example of such a system; as is the third generation UMTS system in TDD mode. Let S denote the number of orthogonal codes used by the base station for transmitting the continuous media streams to the clients. Let $R(j)$, $j = 1, \ldots, J$, denote the number of parallel channels supported by the radio front–end of client j. Let C denote the data rate (in bit/sec) provided by one CDMA code channel in the forward direction.

Our streaming protocol is suitable for any type of prerecorded continuous media. To fix ideas we focus on video streams. A key feature of our protocol is that it accommodates any type of encoding; it accommodates Constant Bit Rate (CBR) and bursty Variable Bit Rate (VBR) encodings as well as encodings with a fixed frame rate (such as MPEG–1 and MPEG–4) and a variable frame rate (such as H.263). For the transmission over the wireless links the video frames are packetized into fixed length packets. The packet size is set such that one CDMA code channel accommodates exactly one packet in one forward slot; thus the base station can transmit S packets on the orthogonal code channels in a forward slot.

Let $N(j),\ j = 1, \ldots, J$, denote the length of the video streams in frames. Let $x_n(j)$ denote the number of packets in the nth frame of video stream j. Let $t_n(j)$ denote the interarrival time between the nth frame and the $(n + 1)$th frame of video stream j in seconds. Frame n is displayed for a frame period of $t_n(j)$ seconds on the client's screen. Because the video streams are prerecorded the sequence of integers $(x_1(j),\ x_2(j), \ldots, x_{N(j)}(j))$ and the sequence of real numbers $(t_1(j),\ t_2(j), \ldots, t_{N(j)}(j))$ are fully known when the streaming commences.

When a client requests a specific video the base station relays the request to the appropriate origin server or proxy server. If the request passes the admission tests the origin/proxy server immediately begins to stream the video via the base station to the client. Our focus in this paper is on the streaming from the base station over the wireless link to the client. We assume for the purpose of this study that the video is delivered to the base station in a timely fashion. Upon granting the client's request the base station immediately commences streaming the video to the client. The packets arriving at the client are placed in the client's prefetch buffer. The video is displayed on the client's monitor as soon as a few frames have arrived at the client. Under normal circumstances the client displays frame n of video stream j for $t_n(j)$ seconds, then removes frame $n + 1$ from its prefetch buffer, decodes it, and displays it for $t_{n+1}(j)$ seconds. If at one of these epochs there is no complete frame in the prefetch buffer the client suffers playback starvation and loses the current frame. At the subsequent epoch the client will attempt to display the next frame of the video.

In our protocol the base station keeps track of the contents of the prefetch buffers in the clients. Toward this end, let $b(j),\ j = 1, \ldots, J$, denote the number of packets in the prefetch buffer of client j. Furthermore, let $p(j),\ j = 1, \ldots, J$, denote the length of the prefetched video segment in the prefetch buffer of client j in seconds. The counters $b(j)$ and $p(j)$ are updated (1) when the client j acknowledges the reception of sent packets, and (2) when a frame is removed, decoded, and displayed at client j.

First, consider the update when packets are acknowledged. For the sake of illustration suppose that the $x_n(j)$ packets of frame n of stream j have been sent to client j during the just expired forward slot. Suppose that all $x_n(j)$ packets are acknowledged during the subsequent backward slot. When the last of the $x_n(j)$ acknowledgments arrives at the base station, the counters are updated by setting $b(j) \leftarrow b(j) + x_n(j)$, and $p(j) \leftarrow p(j) + t_n(j)$.

Next, consider the update of the counters when a frame is removed from the prefetch buffer, decoded, and displayed at the client. Given the sequence $t_n(j),\ n = 1, \ldots, N$, and the starting time of the video playback at the client the base station keeps track of the removal of frames from the prefetch buffer of client j. Suppose that at a particular instant frame $\theta(j)$ is to be removed from the prefetch buffer of client j. The base station tracks the prefetch buffer contents by updating $b(j) \leftarrow [b(j) - x_{\theta(j)}(j)]^+$ and $p(j) \leftarrow [p(j) - t_{\theta(j)}(j)]^+$, where $[x]^+ = \max(0,\ x)$. Note that the client suffers playback starvation when $b(j) - x_{\theta(j)}(j) < 0$, that is, when the frame that is supposed to be removed is not in the prefetch buffer.

3. JSQ PREFETCH POLICY

For each forward slot the base station must decide which packets to transmit from the J ongoing streams. The prefetch policy is the rule that determines which packets are transmitted. The maximum number of packets that can be transmitted in a forward slot is S. The Join–the–Shortest–Queue (JSQ) prefetch policy strives to balance the lengths of the prefetched video segments across all of the clients serviced by the base station. The basic idea is to dynamically assign more codes (and thus transmit more packets in parallel) to clients that have only a small reserve of prefetched video in their prefetch buffers. We first introduce a basic prefetch policy. This basic prefetch policy assumes that all clients (1) support S parallel channels, and (2) have infinite prefetch buffer space. We shall address these two restrictions in a refined prefetch policy.

3.1. Basic JSQ Prefetch Policy

Let $z(j),\ j = 1, \ldots, J$, denote the length of the video segment (in seconds of video run time) that is scheduled for transmission to client j in the current forward slot. The following scheduling procedure is executed for every forward slot. At the beginning of the scheduling procedure all $z(j)$'s are initialized to zero. The base station determines the client j^* with the smallest $p(j) + z(j)$. The base station schedules one packet for transmission (by assigning a code to it) and increments $z(j^*)$: $z(j^*) \leftarrow z(j^*) + t_{\sigma(j^*)}(j^*)/x_{\sigma(j^*)}(j^*)$, where $\sigma(j^*)$ is the frame (number) of stream j^* that is carried (partially) by the scheduled packet. (Although the length of the prefetched video segment grows in increments of $t_n(j)$ seconds whenever the transmission of the $x_n(j)$ packets carrying frame n of stream j is completed; for simplicity we account for partially transmitted frames by incrementing the prefetched segment by $t_n(j)/x_n(j)$ for each transmitted packet.) The base station repeats this procedure S times, that is, until the S available codes are used up. At each iteration the base station determines the j^* with the smallest $p(j) + z(j)$, schedules one packet for client j^* and increments $z(j^*)$. Throughout this scheduling procedure the base station skips packets from a frame that would miss its playback deadline at the client.

During the subsequent backward slot the base station waits for the acknowledgments from the clients. If all packets sent to client j are acknowledged by the end of the backward slot we set $p(j) \leftarrow p(j) + z(j)$. If some of the acknowledgments for a stream j are missing at the end of the backward slot, $p(j)$ is left unchanged. At the end of the backward slot the scheduling procedure starts over. The $z(j)$'s are re–initialized to zero and the base station schedules packets for the clients with the smallest $p(j) + z(j)$.

3.2. Refined JSQ Prefetch Policy

In this section we discuss refinements of the JSQ prefetch policy that limit (1) the number of packets, that are sent (in parallel) to a client in a forward slot, and (2) the number of packets that a client may have in its prefetch buffer. Suppose that the clients j, $j = 1, \ldots, J$, support at most $R(j)$ parallel channels, and have limited prefetch buffer capacities of $B(j)$ packets. Let $r(j)$, $j = 1, .., J$, denote the number of packets scheduled for client j in the upcoming forward slot. Recall that $b(j)$ is the current number of packets in the prefetch buffer of client j. The refinements work as follows. Suppose that the base station is considering scheduling a packet for transmission to client j^*. The base station schedules the packet only if

$$r(j^*) \leq R(j^*) - 1, \qquad (1)$$

and

$$b(j^*) \leq B(j^*) - 1. \qquad (2)$$

If one of these conditions is violated, that is, if the packet would exceed the number of parallel channels of client j^* or the packet would overflow the prefetch buffer of client j^*, the base station removes connection j^* from consideration. The base station next finds a new j^* that minimizes $p(j) + z(j)$. If conditions (1) and (2) hold for the new client j^*, we schedule the packet, update $z(j^*)$ and $r(j^*)$, and continue the procedure of transmitting packets to the clients that minimize $p(j) + z(j)$. Whenever one of the conditions (1) or (2) (or both) is violated we skip the corresponding client and find a new j^*. This procedure stops when we have either (1) scheduled S packets, or (2) skipped over all J streams.

3.3. Simulation of Prefetch Protocol

In this section we describe the simulations of our protocol for continuous media streaming in wireless environments. In our simulations we consider a generic wireless communication system with Time Division Duplex. We assume throughout that the base station allocates $S = 15$ channels (e.g., orthogonal codes in CDMA or time slots in TDMA) to continuous media streaming. We assume that each channel provides a data rate of $C = 64$ kbit/sec in the forward (downlink) direction. Throughout we consider scenarios where all J clients have the same buffer capacity of B packets and support the same number of parallel channels R, i.e., $B(j) = B$ and $R(j) = R$ for all $j = 1, \ldots, J$.

We evaluate our streaming protocol for video streams encoded at a Variable Bit Rate (VBR) and a constant frame rate (e.g., MPEG–1 encodings). We generated ten pseudo traces by scaling MPEG–1 traces obtained from the public domain to an average rate of $\bar{r} = 64$ kbps. The traces have a fixed frame rate of 25 frames/sec and are 40.000 frames long. The generated pseudo traces are highly bursty with peak–to–mean ratios in the range from 7 to 18; see[2] for details.

For each of the J ongoing streams in the wireless cell we select randomly one of the MPEG traces. We generate random starting phases $\theta(j)$, $j = 1, \ldots, J$, into the selected traces. The $\theta(j)$'s are independent and uniformly distributed over the lengths of the selected traces. The frame $\theta(j)$ is removed from the prefetch buffer of client j at

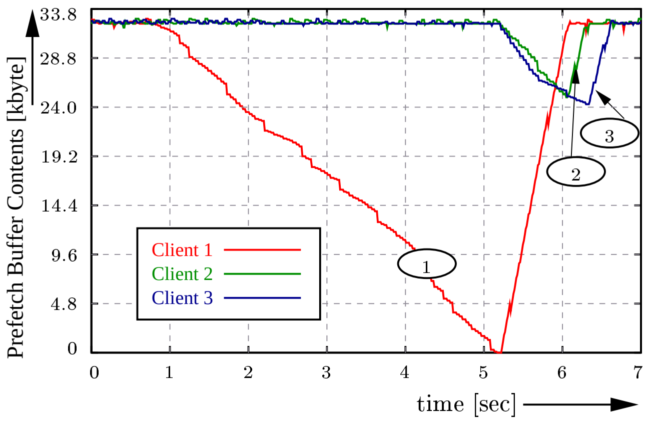

Figure 2. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 starts over

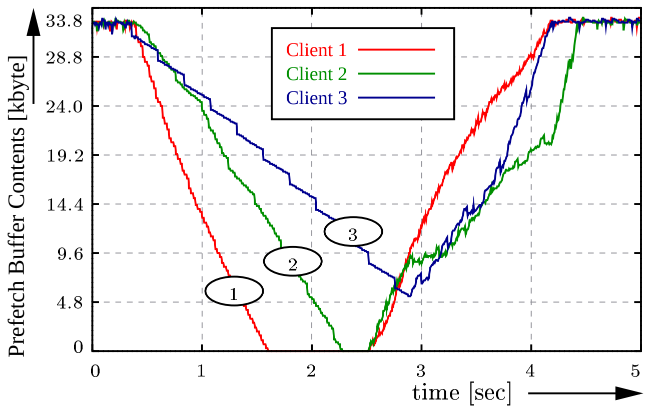

Figure 3. Sample path plot: Prefetch buffer contents (in kBytes) of 3 clients as a function of time: Client 1 experiences a bad channel.

the end of the first frame period. All clients start with empty prefetch buffers. Furthermore, we generate random stream lengths $N(j)$, $j = 1, \ldots, J$. The $N(j)$'s are independent and are drawn from an exponential distribution with mean N_T frames (corresponding to a video run time of T seconds). We initially assume that the client consumes without interruption $N(j)$ frames starting at frame number $\theta(j)$ of the selected trace. When the $N(j)$th frame is removed from the prefetch buffer of client j, we assume that the client immediately requests a new video stream. For the new video stream we again select randomly one of the traces, a new independent random starting phase $\theta(j)$ into the trace, and a new independent random stream lifetime $N(j)$. Thus there are always J streams in progress.

In simulating the wireless links we follow the well–known Gilbert–Elliot model. We simulate each wireless link (consisting of up to $R(j)$ parallel code channels) as an independent discrete–time Markov Chain with two states: "good" and "bad". We assume that all parallel code channels of a wireless link (to a particular client) are either in the good state or the bad state. The transition probabilities of the Markov Chains are set to typical values such that the steady state probability of being in the "good" state is $\pi_g = 0.99$ (and $\pi_b = 0.01$ accordingly); the average sojourn times are 9 seconds for the "good" state and 1 second for the "bad" state. We set the channel error probabilities such that a packet is lost with probability $P_l^g = 0.05$ in the good channel state, and with probability $P_l^b = 1$ in the bad channel state. We assume that acknowledgments are never lost in the simulations.

Figures 2 and 3 show typical sample path plots from the simulations. In this experiment we simulate the streaming to clients with a buffer capacity of $B = 32$ kBytes. The figure shows the prefetch buffer contents of three clients in kBytes. The plots illustrate the collaborative nature of the JSQ prefetch policy in conjunction with the rate adaptation of the wireless communication system. We observe from Figure 2 that at time $t = 5.1$ sec the buffer content of client 1 drops to zero. This is because the video stream of client 1 ends at this time; the client selects a new video stream and starts over with an empty prefetch buffer. Note that already at time $t = 1.0$ second all frames of the "old" video stream have been prefetched into the client's buffer and the client continued to consume frames without receiving any transmissions. When the client starts over with an empty prefetch buffer, the JSQ prefetch policy gives priority to this client and quickly fills its prefetch buffer. While the prefetch buffer of client 1 is being filled the JSQ prefetch policy reduces the transmissions to the other clients; they "live off" their prefetched reserves until client 1 catches up with them.

Notice from Figure 3 that at time $t = 0.4$ sec the buffer occupancy of client 1 drops. This is because this client experiences a bad channel that persists for 2.1 sec (a rather long period chosen for illustration, in our numerical work we set the average sojourn time in the bad channel state to 1 sec), that is, the client is temporarily cut off from the base station. The prefetched reserves allow the client to continue playback during this period. As the prefetched reserves of client 1 dwindle the JSQ prefetch policy allocates larger transmission capacities to it. This, however, cuts down on the transmissions to the other clients, causing their prefetched reserves to dwindle as well. This degrades the performance of the streaming protocol as smaller prefetched reserves make client starvation more likely.

3.4. Channel Probing

In this section we introduce a channel probing refinement designed to improve the performance of the purely JSQ based prefetch protocol. Note that the prefetch protocol introduced in Section 3 does not directly take the physical characteristics of the wireless channels into consideration. The JSQ transmission schedule is based exclusively on the prefetch buffer contents at the clients (and the consumption rates of the video streams). Wireless channels, however, typically experience location–dependent, time–varying, and bursty errors, that is, periods of adverse transmission conditions during which all packets sent to a particular client are lost. Especially detrimental to the prefetch protocol's performance are the persistent bad channels of long–term shadowing that is caused by terrain configuration or obstacles. As illustrated in Figure 3(b), the excessive transmission resources expended on the client that experiences the bad channel, tend to reduce the prefetched reserves of all the other clients in the wireless cell.

To fix this shortcoming we introduce the channel probing refinement. The basic idea is to start *probing* the channel (client) when acknowledgment(s) are missing at the end of a backward slot. While probing the channel the base station sends at most one packet (probing packet) per forward slot to the affected client. The probing continues until an acknowledgment for a probing packet is received. More specifically, if the acknowledgment for at least one packet sent to client j in a forward slot is missing at the end of the subsequent backward slot, we set $R(j) = 1$. This allows the JSQ algorithm to schedule at most one packet (probing packet) for client j in the next forward slot. If the acknowledgment for the probing packet is returned by the end of the next backward slot, we set $R(j)$ back to its original value; otherwise we continue probing with $R(j) = 1$.

To evaluate the performance of this simple probing scheme we compare it with an ideal scenario where the base station has perfect knowledge of the states of the wireless channels. In the perfect knowledge scenario the base station schedules packets only for clients in the good channel state. The base station does not schedule any packet (not even a probing packet) for clients experiencing a bad channel.

3.5. Simulation Results

We now present a quantitative evaluation of the JSQ prefetch protocol. We first define two key measures of the performance of a streaming protocol. We define the *bandwidth efficiency* ξ of a wireless streaming protocol as the sum of the average rates of the streams supported by the base station divided by the total available effective transmission capacity of the base station, i.e.,

$$\xi = \frac{J \cdot \bar{r}}{[\pi_g \cdot (1 - P_l^g) + \pi_b \cdot (1 - P_l^b)] \cdot C \cdot \min(R \cdot J,\ S)}.$$

We define the *client starvation probability* P_{loss} as the long run fraction of encoding information (packets) that misses its playback deadline at the clients. We conservatively consider all $x_n(\cdot)$ packets of frame n as deadline misses when at least one of the frame's packets misses its playback deadline. We warm up each simulation for a period determined with the Schruben's test and obtain confidence intervals on the client starvation probability P_{loss} using the method of batch means. We run the simulations until the 90 % confidence interval of P_{loss} is less than 10 % of its point estimate. Unless stated otherwise, all the following experiments are conducted for the streaming of VBR MPEG video to clients with a buffer capacity of $B = 32$ kBytes and support for $R = 15$ parallel channels. The average lifetime of the video streams is set to $T = 10$ minutes unless stated otherwise. Throughout, the base station has a total of $S = 15$ channels available for video streaming.

Figure 4 shows the client starvation probability P_{loss} without channel probing, with channel probing, and with perfect knowledge of the channel state as a function of the average sojourn time in the "bad" channel state. For this experiment the number of clients is fixed at $J = 13$ (and 12 respectively). We observe from the figure that over a wide range of channel conditions, channel probing is highly effective in reducing the probability of client starvation. In scenarios with persistent bad channel conditions, probing reduces P_{loss} by over one order of magnitude. We also observe that the client starvation probabilities achieved by our simple channel probing scheme are only slightly above the client starvation probabilities achieved with perfect knowledge of the channel state. This indicates that more sophisticated channel probing schemes could achieve only small reductions of the client starvation probability. We set the average sojourn time in the "bad" channel state to 1 sec for all the following experiments.

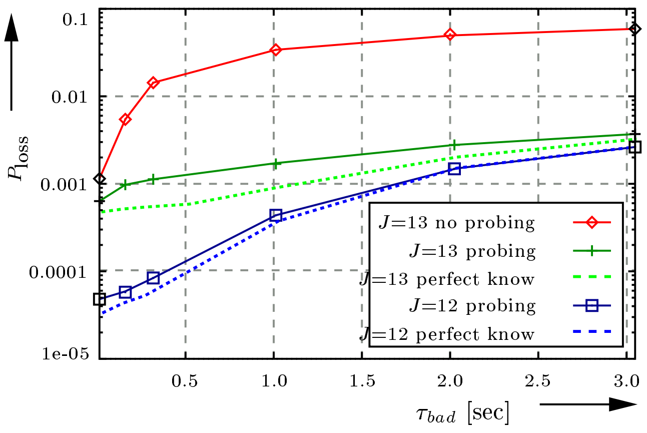

Figure 4. Client starvation probability P_{loss} as a function of the average sojourn time in the "bad" channel state for prefetching of VBR video without channel probing, with channel probing, and with perfect knowledge of the channel state.

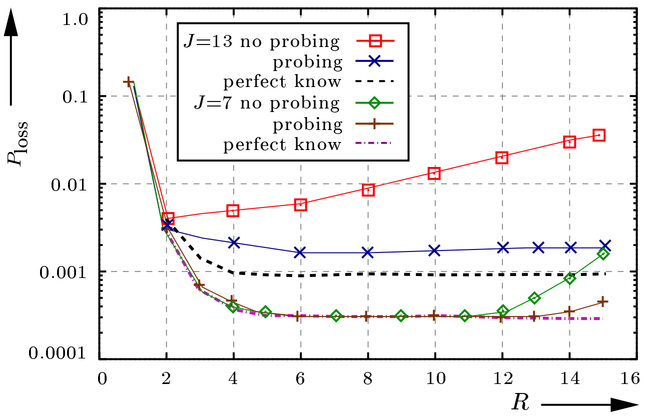

Figure 5. Client starvation probability P_{loss} as a function of the maximum number of channels R per client for prefetching of VBR video without channel probing, with channel probing, and with perfect knowledge of the channel state.

Figure 5 shows the client starvation probability P_{loss} as a function of the maximum number of parallel channels R that can be assigned to an individual client. We observe from Figure 5 that P_{loss} drops by over one order of magnitude as R increases from one to two, allowing for collaborative prefetching through the lending and borrowing of channels. Now consider prefetching with $J = 13$ clients. For prefetching with channel probing and with perfect knowledge of the channel state, P_{loss} drops steadily as R increases. For prefetching without channel probing, however, P_{loss} increases as R grows larger than two. This is because JSQ prefetching without channel probing tends to waste transmission channels on a client experiencing a persistent bad channel. This reduces the prefetched reserves of all clients in the cell, thus increasing the likelihood of client starvation. The larger the number of parallel channels R that can be assigned to an individual client the larger is this waste of transmission channels. Another important observation from Figure 5 is that already a low–cost client with support for a few parallel channels allows for effective prefetching.

Figure 6 shows the client starvation probability P_{loss} as a function of the bandwidth efficiency ξ. Noteworthy is again the effectiveness of the simple channel probing scheme. The client starvation probability P_{loss} achieved with channel probing is (i) generally over one order of magnitude smaller than without channel probing, and (ii) only slightly larger than with perfect knowledge of the channel state. Throughout the remainder of this paper we use prefetching with channel probing. Importantly, the results in Figure 6 indicate that a crude admission control criterion that limits the bandwidth efficiency to less than 0.9, say, is highly effective in ensuring small client starvation probabilities. We note, however, that more research is needed on admission control for streaming in wireless environments.

Figure 7 shows the client starvation probability P_{loss} as a function of the client buffer capacity B. The results demonstrate the dramatic improvement in performance that comes from prefetching. For $J = 13$ ongoing VBR streams the client starvation probability drops by over two orders of magnitude as the clients' buffers increase from 8 kBytes to 128 kBytes. (A buffer of 128 kBytes can hold on average 16 second segments of the VBR videos with an average rate of $\bar{r} = 64$ kbit/sec.) With client buffers of $B = 128$ kBytes and $J = 13$ ongoing streams our prefetch protocol achieves a client starvation probability of less than 10^{-4} and a bandwidth efficiency of 90%!

Table 1 gives the client starvation probability P_{loss} as a function of the average stream lifetime T for the streaming of VBR MPEG–1 video to $J = 13$ clients each with a buffer capacity of $B = 64$ kBytes. Our prefetching protocol performs very well for stream lifetimes on the order of minutes or longer. For stream lifetimes shorter than one minute P_{loss} increases considerably as the lifetime decreases. This is because stream lifetimes this short allow for very little time to build up prefetched reserves. Even for an average stream lifetime of $T = 10$ sec, however, prefetching reduces the client's starvation probability from $2.8{\cdot}10^{-2}$ without any prefetching to $4.5{\cdot}10^{-3}$ with prefetching.

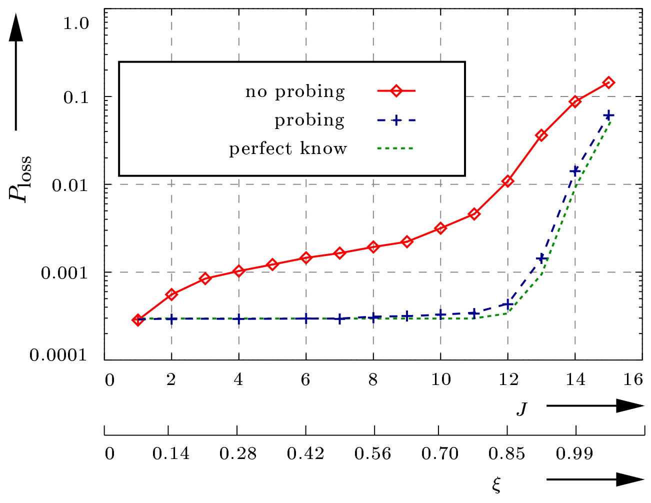

Figure 6. Client starvation probability P_{loss} as a function of the bandwidth efficiency ξ (obtained by varying the number of clients J) for VBR video and prefetch buffer B = 32 kByte.

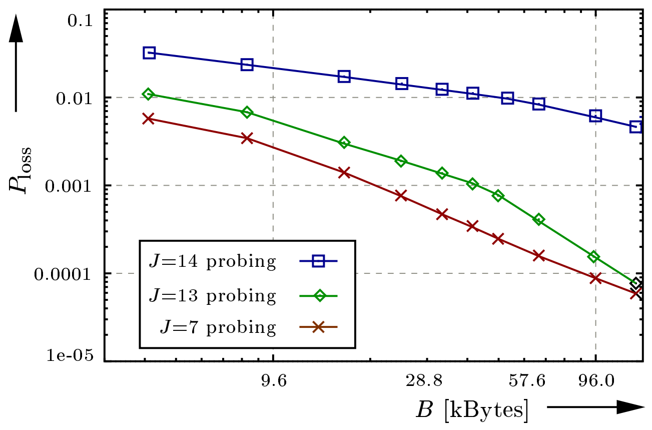

Figure 7. Client starvation probability P_{loss} as a function of the client buffer capacity B for VBR video.

Table 1. Client starvation probability P_{loss} as a function of the average stream lifetime T for VBR video.

T [sec]	10	50	100	600	1200
P_{loss} $[10^{-4}]$	45	7.3	4.6	4.4	2.1

4. RELATED WORK

There is a large body of literature on providing QoS in wireless environments. Much of the work in this area has focused on mechanisms for channel access; see Akyildiz *et al.*[3] for a survey. Choi and Shin[4] have recently proposed a comprehensive channel access and scheduling scheme for supporting real–time traffic and non–real–time traffic on the uplinks and downlinks of a wireless LAN. Recently, packet fair scheduling algorithms that guarantee clients a fair portion of the shared transmission capacity have received a great deal of attention.[5–7] These works adapt fair scheduling algorithms originally developed for wireline packet–switched networks to wireless environments. Another line of work addresses the efficiency of reliable data transfer over wireless links.[8,9]

We note that to our knowledge *none of the existing schemes for providing QoS in wireless environments takes advantage of the special properties (predictability and prefetchability) of prerecorded continuous media*, that are expected to account for a large portion of the future Internet traffic. There is an extensive literature on the streaming of prerecorded continuous media, in particular VBR video, over *wireline* packet–switched networks; see Krunz[10] as well as Reisslein and Ross[11] for a survey. In this literature a wide variety of smoothing and prefetching schemes is explored to efficiently accommodate VBR video on fixed bandwidth wireline links. Among these schemes is a prefetching scheme based on the Join–the–Shortest–Queue (JSQ) principle developed by Reisslein and Ross.[11] Their scheme is designed for a Video on Demand service with VBR encoded fixed frame rate MPEG video over an ADSL network or the cable plant. It does not handle the location–dependent, time–varying, and bursty errors that are typical for wireless environments.

Elaoud and Ramanathan[12] propose a scheme for providing network level QoS to flows in a wireless CDMA system. Their scheme dynamically adjust the signal to interference and noise ratio requirements of flows based on MAC packet deadlines and channel conditions. The Simultaneous MAC Packet Transmission (SMPT) scheme of Fitzek *et al.*[13] provides transport level QoS by exploiting rate adaptation techniques of CDMA systems. The SMPT scheme delivers transport layer segments (e.g., UDP or TCP segments, which are divided into several MAC packets) with high probability within a permissible delay bound. Our work in this paper differs from[12,13] in that they propose decentralized schemes for backward (uplink) transmissions. Also, there is no prefetching in.[12,13] Moreover,[12,13] do not take the characteristics of the application layer traffic into consideration; the scheme[12] operates on one MAC packet at a time and SMPT[13] operates on one TCP or UDP segment at a time.

5. CONCLUSION

We have developed a high performance prefetching protocol for the streaming of prerecorded continuous media in a cellular wireless system. Our prefetching protocol can be employed on top of any of the rate adaptation techniques of wireless communication systems. Our protocol accommodates CBR and VBR encodings as well as fixed frame rate and variable frame rate encodings. Channel probing is critical for the performance of our protocol. With channel probing the base station allocates transmission resources in a judicious manner avoiding the allocation of large portions of the available transmission capacity to clients experiencing adverse transmission conditions.

In the extended technical report[2] we conduct a more extensive evaluation and discuss several extensions of the prefetching protocol. We consider streaming with client interactions, such as pause/resume and temporal jumps. We also consider the streaming of live content, such as the audio and video feed from a sporting event. We outline how to use the prefetching protocol for prefetching in the uplink (clients to base station) direction. Moreover, we outline how to deploy the prefetching protocol in third generation CDMA systems as well as TDMA and FDMA systems. We also discuss the deployment of the prefetching protocol on top of physical layer error control schemes.

Acknowledgment: We are grateful to Prof. Adam Wolisz for providing the environment that allowed us to pursue the work presented in this paper.

REFERENCES

1. S. Nanda, K. Balachandran, and S. Kumar, "Adaptation techniques in wireless packet data services," *IEEE Communications Magazine* **38**, pp. 54–64, Jan. 2000.
2. F. Fitzek and M. Reisslein, "A prefetching protocol for continuous media streaming in wireless environments (extended version)," Tech. Rep. TKN–00–05, Technical University Berlin, Dept. of Electrical Eng., Germany, Dec. 2000. available at `http://www-tkn.ee.tu-berlin.de/~fitzek` and `http://www.eas.asu.edu/~mre`.
3. I. F. Akyildiz, J. McNair, L. C. Martorell, R. Puigjaner, and Y. Yesha, "Medium access control protocols for multimedia traffic in wireless networks," *IEEE Network* **13**, pp. 39–47, July/August 1999.
4. S. Choi and K. G. Shin, "A unified wireless LAN architecture for real–time and non–real–time communication services," *IEEE/ACM Transactions on Networking* **8**, pp. 44–59, Feb. 2000.
5. C. Fragouli, V. Sivaraman, and M. B. Srivastava, "Controlled multimedia wireless link sharing via enhanced class–based queueing with channel–state–dependent packet scheduling," in *Proceedings of IEEE Infocom '98*, (San Francisco, CA), Apr. 1998.
6. T. S. E. Ng, I. Stoica, and H. Zhang, "Packet fair queueing algorithms for wireless networks with location dependent errors," in *Proceedings of IEEE Infocom '98*, pp. 1103–1111, (San Francisco, CA), Apr. 1998.
7. S. Lu, T. Nandagopal, and V. Bharghavan, "A wireless fair service algorithm for packet cellular networks," in *Proceedings of ACM/IEEE MobiCom '98*, pp. 10–20, (Dallas, TX), Oct. 1998.
8. S. S. E. Amir, H. Balakrishnan and R. Katz, "Efficient TCP over networks with wireless links," in *Proceedings of ACM Conference on Mobile Computing and Networking*, (Berkeley, CA), Dec. 1995.
9. P. Bhagwat, P. Bhattacharya, A. Krishna, and S. K. Tripathi, "Using channel state dependent packet scheduling to improve TCP throughput over wireless LANs," *ACM Wireless Networks* **3**, pp. 91–102, 1997.
10. M. Krunz, "Bandwidth allocation strategies for transporting variable–bit–rate video traffic," *IEEE Communications Magazine* **37**, pp. 40–46, Jan. 1999.
11. M. Reisslein and K. W. Ross, "High–performance prefetching protocols for VBR prerecorded video," *IEEE Network* **12**, pp. 46–55, Nov/Dec 1998.
12. M. Elaoud and P. Ramanathan, "Adaptive allocation of CDMA resources for network–level QoS assurance," in *Proceedings of ACM MobiCom 2000*, (Boston, MA), Aug. 2000.
13. F. H. P. Fitzek, B. Rathke, M. Schlager, and A. Wolisz, "Quality of service support for real–time multimedia applications over wireless links using the simultaneous MAC–packet transmission (SMPT) in a CDMA environment," in *Proceedings of 5th International Workshop on Mobile Multimedia Communications (MoMuC)*, pp. 367–378, (Berlin, Germany), Oct. 1998.